UNITED STATES PATENT APPLICATION

FOR

Broadcast State Renaming in a Microprocessor

INVENTORS:

Michael P. Cornaby
Lawrence O. Smith

Prepared by:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CALIFORNIA 90025
(408) 720-8300

Attorney's Docket No. 42390P12007

Broadcast State Renaming in a Microprocessor

FIELD OF THE INVENTION

[0001]   The present invention relates generally to the field of microprocessors and, more specifically, to a method and apparatus for renaming broadcast state in a microprocessor.

BACKGROUND OF THE INVENTION

[0002]   In a microprocessor, "state" may be defined as the value, at a given point in time, of one or more parameters used by the microprocessor in performing its functions. For example, microprocessors designed according to the Intel Architecture typically maintain a set of hardware registers, known as the "segment register file", which typically are located in the memory execution unit (MEU) of the microprocessor. Two of these registers, the code segment register and the stack segment register, have particular importance regarding processor "state". The purpose of the code segment register is to provide protection information and basic operating parameters as applied to instruction fetch (e.g., location in memory from which an instruction can be fetched, privilege levels). The purpose of the stack segment register is to indicate areas in memory where programs can read and/or write stack information, privilege levels, etc. Each of these two registers (and other registers) contains multiple bits, collectively representing the values of several different processor parameters. The term "state" commonly refers to the values of any one or more of these parameters.

1

[0003]　A typical microprocessor includes a number of different functional units, each of which requires current state (i.e., the value of one or more of the afore-mentioned parameters) to perform its functions during the processing of microinstructions. Examples of such functional units in the Intel Architecture are the microinstruction checker retirement unit (CRU), instruction translation look-aside buffer (ITLB), trace cache (TC), and segment and address translation unit (SAAT). Consequently, such microprocessors commonly use a process known as state "broadcasting", in which the current state is distributed, via one or more internal buses, to the various functional units which require it. State frequently changes during operation of the microprocessor, and it must be rebroadcast to the various functional units when that occurs. State tends to change frequently in a microprocessor, such as in response to any branch in the macrocode (a "far transfer") and, when in protected mode, in response to any architectural event.

[0004]　Broadcasting of state consumes microinstructions and therefore tends to have a negative performance impact on the processor, i.e., it slows down the processor. Broadcasting of state can have a more significant negative impact when used in conjunction with certain performance-enhancing technologies, such as out-of-order execution. Out-of-order execution is a well-known technique used in pipelined microprocessors to improve instruction throughput. Out-of-order execution involves selectively executing certain instructions out of order, i.e., before other instructions that occur "earlier" in the program code flow. A technique called register renaming is often used to facilitate out-of-order execution. Register

2

renaming allows multiple execution paths without conflicts between different execution units trying to use the same registers. Rather than just one set of registers being used, multiple sets of registers are provided in the processor. This allows different execution units to work simultaneously without unnecessary pipeline stalls.

[0005] A problem associated with register renaming, however, is that it tends to increase the frequency with which state must be rebroadcast, degrading processor performance. In current microprocessors using the Intel Architecture, each functional unit stores a single copy of the state it requires (i.e., a portion or all of the code segment and/or stack segment registers). Consequently, whenever the code segment or stack segment register is renamed, state must be rebroadcast to the various functional units that require it. However, the rebroadcast cannot take place until all of the microinstructions which are based on the old state have finished executing. As a result, the microinstruction pipeline must stall until that happens, which slows down the processor. Thus, the requirement of having to frequently rebroadcast state tends to degrade performance of a microprocessor.

[0006] What is needed, therefore, is a technique which addresses the performance problems associated with rebroadcasting of state, particularly in an out-of-order microprocessor that uses register renaming.

3

SUMMARY OF THE INVENTION

[0007]   The present invention includes a method and apparatus for storing a plurality of copies of broadcast state of a processor in each of multiple functional units of the processor. A microinstruction including an identifier is issued in the processor, and the identifier is used in at least one of the functional units to select one of the copies of broadcast state for use in processing the microinstruction.

[0008]   Other features of the present invention will be apparent from the accompanying drawings and from the detailed description which follows.

4

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] The present invention is illustrated by way of example and not limited in

the figures of the accompanying drawings, in which like references indicate similar

elements, and in which:

[0010] Figure 1 is a block diagram illustrating one embodiment of a pipeline of a

processor;

[0011] Figure 2 is a block diagram illustrating an architecture of a microprocessor

in which the present invention is implemented;

[0012] Figure 3 shows a functional unit of a microprocessor containing multiple

copies of broadcast state;

[0013] Figure 4 shows a microinstruction including additional modifiers for use in

broadcast state renaming; and

[0014] Figure 5 shows a broadcast state machine implemented in the microcode

sequencer.

<u>DETAILED DESCRIPTION</u>

[0015]   A method and apparatus for renaming broadcast state in a microprocessor are described. Note that in this description, references to "one embodiment" or "an embodiment" mean that the feature being referred to is included in at least one embodiment of the present invention. Further, separate references to "one embodiment" in this description do not necessarily refer to the same embodiment; however, neither are such embodiments mutually exclusive, unless so stated and except as will be readily apparent to those skilled in the art. For example, a feature, structure, act, etc. described in one embodiment may also be included in other embodiments. Thus, the present invention can include a variety of combinations and/or integrations of the embodiments described herein.

I. Overview

[0016]   As described in greater detail below, a processor uses broadcast state renaming to reduce processing delays and microcode overhead which would otherwise result from having to rebroadcast state (such as in response to register renaming). The processor includes a memory execution unit, a microcode sequencer, and various functional units. The memory execution unit includes segment registers, content of which represent state of the processor. The microcode sequencer sets an identifier field in at least some microinstructions, indicating which of multiple copies of broadcast state are to be used for processing each microinstruction. Each functional unit which requires broadcast state receives and internally stores multiple copies of broadcast state. Each copy of broadcast state

6

may correspond to a different renamed version of a segment register. When processing a microinstruction, each functional unit selects, based on the identifier field of the microinstruction, one of its internally stored copies of broadcast state for use in processing the microinstruction.

[0017] Among other advantages, this technique eliminates the need, when a rebroadcast is required, to wait for all previous microinstructions that use broadcast state to finish executing before issuing the rebroadcast. In addition, this technique reduces the need for rebroadcasts and reduces microcode overhead otherwise attributable to such rebroadcasts. Consequently, this technique improves the overall processing speed and performance of the processor.

[0018] Before describing the broadcast state renaming technique in detail (see under heading "II. Broadcasting State ", below), it is useful to consider the architecture of a microprocessor in which the technique can be implemented. Figure 1 is a high-level block diagram illustrating one embodiment of a processor pipeline 10 used in such a processor. The pipeline 10 includes a number of pipeline stages, commencing with a fetch stage 12 at which instructions (e.g., macroinstructions) are retrieved and fed into the pipeline 10. For example, a macroinstruction may be retrieved from a cache memory that is integral with the processor, or closely associated with it, or may be retrieved from an external main memory via a processor bus. From the fetch stage 12, the macroinstructions are propagated to a decode stage 14, where macroinstructions are translated into microinstructions (also termed "microcode") suitable for execution within the

processor. The microinstructions are then propagated downstream to an allocate stage 16, where processor resources are allocated to the various microinstructions according to availability and need. The microinstructions are then executed at an execute 18 before being retired, or "written-back" (e.g., committed to an architectural state) at a retire stage 20.

[0019]   Figure 2 is a block diagram illustrating an embodiment of a general-purpose programmable microprocessor ("processor") 30 that may implement pipeline 10. The processor 30 is described herein as being a multithreaded (MT) processor, and is accordingly able to process multiple instruction threads (or contexts). However, many of the teachings provided herein are not specific to an MT processor, and may find application in a single-threaded processor. In one embodiment, the processor 30 may comprise an Intel Architecture (IA) microprocessor capable of executing the IA instruction set, such as manufactured by Intel Corporation of Santa Clara, California.

[0020]   The processor is capable of out-of-order execution of instructions. Accordingly, in one embodiment the processor 30 includes an in-order front end and an out-of-order back end. The in-order front end includes a bus interface unit 32, which functions as the conduit between the processor 30 and other components (e.g., main memory) of a computer system within which the processor 30 may be used. To this end, the bus interface unit 32 couples the processor 30 to a processor bus (not shown) via which data and control information may be received at, and propagated from, the processor 30. The bus interface unit 32 receives bus requests

8

38 from, and sends snoops or bus returns to, a memory execution unit (MEU) 42

that provides local memory capability within the processor 30. The MEU 42

includes a unified data and instruction cache 44, a data Translation Lookaside

Buffer (TLB) 46, and memory ordering buffer 48. The memory execution unit 42

receives instruction fetch requests 50 from, and delivers raw instructions 52 (i.e.,

coded macroinstructions) to, a microinstruction translation engine 54 that translates

the macroinstructions into a corresponding set of microinstructions.

[0021]   The MEU 42 also includes a segment and address translation unit (SAAT)

33 and a register file 34. The register file 34 includes a number of special-purpose

registers used by the processor 30, including a code segment register (CSR) 35 and a

stack segment register (SSR) 36. Values stored in the CSR 35 and SSR 36 represent

state of the processor 30, which is broadcast to various functional units of the

processor 30 at various times.

[0022]   The microinstruction translation engine 54 effectively operates as a trace

cache "miss handler", in that it operates to deliver microinstructions to a trace cache

62 in the event of a trace cache miss. To this end, the microinstruction translation

engine 54 functions to provide the fetch and decode pipe stages 12 and 14 in the

event of a trace cache miss. The microinstruction translation engine 54 is shown to

include a next instruction pointer (NIP) 100, an instruction Translation Lookaside

Buffer (ITLB) 102, a branch predictor 104, an instruction streaming buffer 106, an

instruction pre-decoder 108, instruction steering logic 110, an instruction decoder

112, and a branch address calculator 114. The next instruction pointer 100, ITLB

9

102, branch predictor 104 and instruction streaming buffer 106 together constitute a branch prediction unit (BPU). The instruction decoder 112 and branch address calculator 114 together comprise an instruction translate (IX) unit.

[0023] The next instruction pointer 100 issues next instruction requests to the unified cache 44. In the exemplary embodiment where the processor 30 comprises a multithreaded microprocessor capable of processing two threads, the next instruction pointer 100 may include a multiplexer ("mux") (not shown) that selects between instruction pointers associated with either the first or second thread for inclusion within the next instruction request issued therefrom. In one embodiment, the next instruction pointer 100 will interleave next instruction requests for the first and second threads on a cycle-by-cycle ("ping pong") basis, assuming instructions for both threads have been requested, and instruction streaming buffer 106 resources for both of the threads have not been exhausted. The next instruction pointer requests may be for 16, 32 or 64-bytes depending on whether the initial request address is in the upper half of a 32-byte or 64-byte aligned line. The next instruction pointer 100 may be redirected by the branch predictor 104, the branch address calculator 114 or by the trace cache 62, with a trace cache miss request being the highest priority redirection request.

[0024] When the next instruction pointer 100 makes an instruction request to the unified cache 44, it generates a two-bit "request identifier" that is associated with the instruction request and functions as a "tag" for the relevant instruction request. When returning data responsive to an instruction request, the unified cache 44

returns the following tags or identifiers together with the data:

1.   The "request identifier" supplied by the next instruction pointer 100;

2.   A three-bit "chunk identifier" that identifies the chunk returned; and

3.   A "thread identifier" that identifies the thread to which the returned data belongs.

[0025]   Next instruction requests are propagated from the next instruction pointer 100 to the instruction ITLB 102, which performs an address lookup operation, and delivers a physical address to the unified cache 44. The unified cache 44 delivers a corresponding macroinstruction to the instruction streaming buffer 106. Each next instruction request is also propagated directly from the next instruction pointer 100 to the instruction streaming buffer 106 so as to allow the instruction streaming buffer 106 to identify the thread to which a macroinstruction received from the unified cache 44 belongs. The macroinstructions from both first and second threads are then issued from the instruction streaming buffer 106 to the instruction pre-decoder 108, which performs a number of length calculation and byte marking operations with respect to a received instruction stream (of macroinstructions). Specifically, the instruction pre-decoder 108 generates a series of byte marking vectors that serve, among other purposes, to demarcate macroinstructions within the instruction stream propagated to the instruction steering logic 110.

[0026]   The instruction steering logic 110 then utilizes the byte marking vectors to steer discrete macroinstructions to the instruction decoder 112 for the purposes of decoding. Macroinstructions are also propagated from the instruction steering logic

11

110 to the branch address calculator 114 for the purposes of branch address calculation. Microinstructions are then delivered from the instruction decoder 112 to the trace delivery engine 60.

[0027]   During decoding, flow markers are associated with each microinstruction into which a macroinstruction is translated. A flow marker indicates a characteristic of the associated microinstruction and may, for example, indicate the associated microinstruction as being the first or last microinstruction in a microcode sequence representing a macroinstruction. The flow markers include a "beginning of macroinstruction" (BOM) and an "end of macroinstruction" (EOM) flow markers. According to the present invention, the decoder 112 may further decode the microinstructions to have shared resource (multiprocessor) (SHRMP) flow markers and synchronization (SYNC) flow markers associated therewith. Specifically, a shared resource flow marker identifies a microinstruction as a location within a particular thread at which the thread may be interrupted (e.g., re-started or paused) with less negative consequences than elsewhere in the thread. The decoder 112, in an exemplary embodiment of the present invention, is constructed to mark microinstructions that comprise the end or the beginning of a parent macroinstruction with a shared resource flow marker as well as intermittent points in longer microcode sequences. A synchronization flow marker identifies a microinstruction as a location within a particular thread at which the thread may be synchronized with another thread responsive to, for example, a synchronization instruction within the other thread.

12

[0028]   The decoder 112, in one embodiment, is constructed to mark microinstructions that are located at selected macroinstruction boundaries where state shared among threads coexisting in the same processor can be changed by one thread without adversely impacting the execution of other threads.

[0029]   From the microinstruction translation engine 54, decoded instructions (i.e., microinstructions) are sent to a trace delivery engine 60. The trace delivery engine 60 includes a trace cache 62, a trace branch predictor (BTB) 64, a microcode sequencer 66 and a microcode (uop) queue 68. The trace delivery engine 60 functions as a microinstruction cache, and is the primary source of microinstructions for a downstream execution unit 70. By providing a microinstruction caching function within the processor pipeline, the trace delivery engine 60, and specifically the trace cache 62, allows translation work done by the microinstruction translation engine 54 to be leveraged to provide increased microinstruction bandwidth. In one embodiment, the trace cache 62 may comprise a 256 set, 8 way set associate memory. The term "trace" refers to a sequence of microinstructions stored within entries of the trace cache 62, each entry including pointers to preceding and proceeding microinstructions comprising the trace. In this way, the trace cache 62 facilitates high-performance sequencing in that the address of the next entry to be accessed for the purposes of obtaining a subsequent microinstruction is known before a current access is complete. Traces may be viewed as "blocks" of instructions that are distinguished from one another by trace heads, and are terminated upon encountering an indirect branch or by reaching one

13

of many present threshold conditions, such as the number of conditioned branches that may be accommodated in a single trace or the maximum number of total microinstructions that may comprise a trace.

[0030] The trace cache branch predictor 64 provides local branch predictions pertaining to traces within the trace cache 62. The trace cache 62 and the microcode sequencer 66 provide microinstructions to the microcode queue 68, from where the microinstructions are then fed to an out-of-order execution cluster. The microcode sequencer 66 includes a number of event handlers 67, embodied in microcode, that implement a number of operations within the processor 30 in response to the occurrence of an event such as an exception or an interrupt. The event handlers 67, as will be described in further detail below, are invoked by an event detector 188 included within a register renamer 74 in the back end of the processor 30.

[0031] The microcode sequencer 66 also includes a broadcast state machine 69, which is described further below. The broadcast state machine 69 controls which of multiple copies of broadcast state are to be used at any given time by the various functional units of the processor 30.

[0032] The processor 30 may be viewed as having an in-order front-end, comprising the bus interface unit 32, the memory execution unit 42, the microinstruction translation engine 54 and the trace delivery engine 60, and an out-of-order back-end that will be described in detail below.

[0033] Microinstructions dispatched from the microcode queue 68 are received into an out-of-order cluster 71 comprising a scheduler 72, a register renamer 74, an

14

allocator 76, a microinstruction checker retirement unit (CRU) 79 containing a reorder buffer 78, and a replay queue 80. The scheduler 72 includes a set of reservation stations, and operates to schedule and dispatch microinstructions for execution by the execution unit 70. The register renamer 74 performs a register renaming function with respect to hidden integer and floating point registers (that may be utilized in place of any of the eight general purpose registers or any of the eight floating-point registers, where a processor 30 executes the Intel Architecture instruction set). The allocator 76 operates to allocate resources of the execution unit 70 and the cluster 71 to microinstructions according to availability and need. In the event that insufficient resources are available to process a microinstruction, the allocator 76 is responsible for asserting a stall signal 82, that is propagated through the trace delivery engine 60 to the microinstruction translation engine 54, as shown at 58. Microinstructions, which have had their source fields adjusted by the register renamer 74, are placed in a reorder buffer 78 in strict program order. When microinstructions within the reorder buffer 78 have completed execution and are ready for retirement, they are then removed from a reorder buffer and retrieved in an in-order manner (i.e., according to an original program order). The replay queue 80 propagates microinstructions that are to be replayed to the execution unit 70.

[0034]   The execution unit 70 includes a floating-point execution engine 84, an integer execution engine 86, and a level 0 data cache 88. In one exemplary embodiment in which is the processor 30 executes the Intel Architecture instruction set, the floating point execution engine 84 may further execute MMX® instructions

15

and Streaming SIMD (Single Instruction, Multiple Data) Extensions (SSEs).

II. Broadcasting State

[0035]   The processor 30 includes a number of functional units, as described above, such as SAAT 33, ITLB 102, trace cache 62, and CRU 79. Many of these functional units may use broadcast state when processing microinstructions. State is broadcast from the execution unit 70 to the various functional units that require it, via internal buses or other connections within the processor 30.

[0036]   In accordance with the present invention, each functional unit in processor 30 that requires broadcast state (including any of the aforementioned functional units) receives and internally stores multiple different copies of broadcast state. Figure 3 shows an abstraction of a functional unit 301, representing any functional unit in processor 30 which uses broadcast state. The functional unit 301 includes logic 302 for performing its intended functions. In addition, the functional unit 301 includes multiple different copies 303 of broadcast state, i.e., state copy 0 and state copy 1 in this example, which may be stored within registers in the functional unit 301. Note that in other embodiments, functional units may maintain more than two different copies of broadcast state. However, in this description is assumed that each functional unit maintains two copies, to simplify description.

[0037]   Figure 4 shows an example of the format of a microinstruction to allow broadcast state renaming in accordance with the present invention. As shown, a microinstruction 400 in accordance with the present invention includes a standard uop code 401, a first source field (operand) 402, and a second source field (operand)

16

403. In addition, the microinstruction 400 includes three additional modifiers for use in broadcast state renaming: a Bcast field 404, a Bcast_State_Change field 405, and a First_Bcast_State_Use field 406.

[0038]   A Bcast field 404 is included in every microinstruction and indicates to the various functional units which of their multiple internally-stored copies of broadcast state to use when processing that microinstruction. In an embodiment in which only two copies of broadcast state are maintained by each functional unit, the Bcast field 404 may be a single bit ("the Bcast bit"), where each possible value of the Bcast bit corresponds to a different copy of broadcast state. The Bcast field 404 is set by the broadcast state machine 69 in the microcode sequencer 66 in the manner described below, based on the values of the Bcast_State_Change field 405 and the First_Bcast_State_Use field 406.

[0039]   The Bcast_State_Change field 405 signals the microcode sequencer 66 that the microinstruction will cause a broadcast state change. The First_Bcast_State_Use field 406 indicates that the microinstruction is the first user of a new broadcast state and, therefore, indicates to the microcode sequencer 66 at which microinstruction to stall until a new broadcast state is received and propagated. The Bcast_State_Change field 405 and the First_Bcast_State_Use field 406 are determined by the microcode programmer.

[0040]   The microcode sequencer 66 stalls whenever the number of state changes in flight equals the number of copies of state maintained by each functional unit. Thus, if the Bcast field 404 is a single-bit field (i.e., two copies of state in each

17

functional unit), the microcode sequencer 66 stalls whenever another broadcast-changing uop (microinstruction) attempts to issue while a broadcast change is in progress. In addition, on every mispredicted macrobranch or microbranch ("branch mispredict"), the execution unit 70 sends the Bcast bit associated with the mispredicted uop to the microcode sequencer 66. The trace cache 62 also sends the Bcast bit to the execution unit 70 for every uop which can use broadcast state and with every mispredictable macrobranch.

[0041]   Figure 5 shows an embodiment of the broadcast state machine 69 implemented in the microcode sequencer 66, which determines the state of the Bcast field 404 (in this case, the Bcast bit). In state 501, the Bcast bit is equal to 0. When any uop that has its Bcast bit set (a "broadcast uop") issues while the state machine 69 is in state 501, the state machine 69 enters 0-to-1 transition state 502, in which the Bcast bit is about to be set to 1. While in transition state 502, if a branch mispredicts occurs with the Bcast bit equal to 1, there is no change in state. If another broadcast uop issues while in transition state 502, the state machine 69 enters stall state 505, in which microinstruction execution is stalled. While in transition state 502, if a branch mispredicts occurs with the Bcast bit equal to 0, i.e. a "nuke", the state machine 69 transitions back to state 501. A "nuke" may be defined as a state clearing operation which reverts all of the speculative updates in the processor back to non-speculative state. While in transition state 502, if all uops for the broadcast have been retired (some broadcasts may require multiple uops), the state machine 69 transitions to state 503, in which the Bcast bit is set to one.

18

[0042] While in stall state 505, any branch mispredict with the Bcast bit equal to 1 causes no change in state. However, if a branch mispredicts occurs with the Bcast bit equal to 0 while in stall state 505 (a nuke), the state machine 69 transitions back to state 501. If all uops for the broadcast have been retired while in stall state 505, the state machine 69 transitions to state 503, in which the Bcast bit is set to 1.

[0043] While in state 503, when a broadcast uop issues, the state machine 69 enters 1-to-0 transition state 504, in which the Bcast bit is about to be set to 0. While in transition state 504, if a branch mispredicts occurs with the Bcast bit equal to 0, there is no change in state. If another broadcast uop issues while in transition state 504, the state machine 69 enters stall state 506, in which microinstruction execution is stalled. While in transition state 504, if a branch mispredicts occurs with the Bcast bit equal to 1 (a nuke), the state machine 69 transitions back to state 503. While in transition state 504, if all uops for the broadcast have been retired, the state machine 69 transitions to state 501, in which the Bcast bit is set to 0.

[0044] While in stall state 506, any branch mispredict with the Bcast bit equal to 0 causes no change in state. However, if a branch mispredicts occurs with the Bcast bit equal to 1 while in stall state 506 (a nuke), the state machine 69 transitions back to state 503. If all uops for the broadcast have been retired while in stall state 506, the state machine 69 transitions to state 501, in which the Bcast bit is set to 0.

[0045] To implement more than two copies of broadcast state, the CRU 79 can maintain a retirement version of the Bcast field 404 for an event restart, and the microcode sequencer 66 can track the Bcast field allocation as well as the allocation

19

order for misprediction recoveries.

[0046]   Thus, a method and apparatus for renaming broadcast state in a microprocessor have been described. Although the present has been described with reference to specific exemplary embodiments, it will be evident that various modifications and changes may be made to these embodiments without departing from the broader scope and spirit of the invention. Accordingly, the specification and drawings are to be regarded in an illustrative rather than a restrictive sense.